A Tool for Defining Agent Protocols in HAPN (Demonstration)

Nitin Yadav
School of Computer Science and I.T.
RMIT University
Melbourne, Australia
nitin.yadav@rmit.edu.au

Lin Padgham
School of Computer Science and I.T.
RMIT University
Melbourne, Australia
lin.padgham@rmit.edu.au

Michael Winikoff
Department of Information Science
University of Otago
Dunedin, New Zealand
michael.winikoff@otago.ac.nz

ABSTRACT

This demonstration is exhibiting an interactive tool for defining agent protocols using a new notation "HAPN" which we have developed to overcome issues we have experienced with commonly used agent protocol notations such as AUML. The notation has a formal semantics which facilitates back end support within the tool for checking desirable or undesirable properties of a specification. The notation is an extension of hierarchical finite state machines and the tool is implemented in HTML5 and Javascript.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence — Multiagent systems

Keywords

Communication languages and protocols, Modelling and specification languages

1. INTRODUCTION

In designing multi-agent systems, interaction protocols are key to defining agent communication patterns. In this demonstration, we will showcase a graphical editor that can be used to design protocols using a new notation that we call Hierarchical Agent Protocol Notation (HAPN).

In our experience, we have found existing protocol notations to be problematic in designing, developing and teaching multi-agent systems. Existing *graphical* notations, of which Agent UML (AUML) is arguably the most popular, in our experience are problematic as they are difficult to use correctly and are unable to model certain key aspects of interactions. The overarching aim behind HAPN is to provide a *pragmatic* notation that is unambiguous and easy to use by protocol designers without losing preciseness or the ability to formally verify and validate protocols developed using it.

HAPN is based on hierarchical finite state machines [1] as the underlying conceptual framework. This allows defining interaction protocols *graphically*, using a simple and accessible notation, while retaining a formal structure hidden from the designer (that can be used by the editor for formal analysis behind the scenes).

In the next section, we briefly introduce basic syntax of the notation and how to define protocols using it in the graphical editor.

Appears in: *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015), Bordini, Elkind, Weiss, Yolum (eds.), May 4–8, 2015, Istanbul, Turkey.*

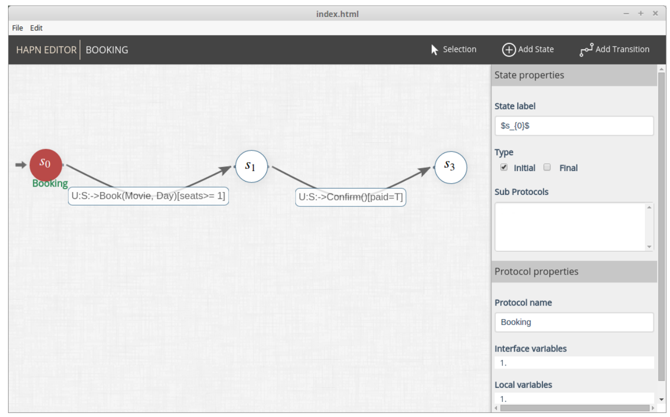

Figure 1: HAPN-E: Graphical editor for HAPN.

2. GRAPHICAL EDITOR FOR HAPN

A protocol in our notation is an extension of Finite State Machines (FSMs) built using using messages, guards and effects to form transitions between states. A transition in HAPN has the structure $\mathsf{msg}[\mathsf{guards}]/\mathsf{effects}$ where msg is a message, guards are conditions on the message, and $\mathsf{effects}$ are effects of the transition. A *message* is defined between two roles and consists of a message name and (optional) arguments.

The graphical editor HAPN-E (shown in Figure 1) has two key areas, the left region serves as a *canvas* where a designer will layout various elements of our notation, namely, protocols, states, and transitions; and the right region displays property forms to modify these elements. The HAPN editor has three key operation modes, Selection, Add State, and Add Transition, which can be activated by clicking the buttons of same name (see Figure 1). The Add State and Add Transition modes allow adding states and transitions on the canvas, whereas the selection mode allows selecting existing states and transitions to modify their respective properties.

2.1 States

A new state can be added by selecting the Add State mode and clicking on the canvas. Clicking on a state in the selection mode displays a form to modify its properties (see property form in Figure 1). The editor toggles the visibility of relevant property sections based on the user's interaction. For example, a protocol properties form is made visible automatically when a user marks a state as initial.

We show the protocol name below its initial state, for instance the protocol name "Booking" is shown below initial state s_0 in Fig-

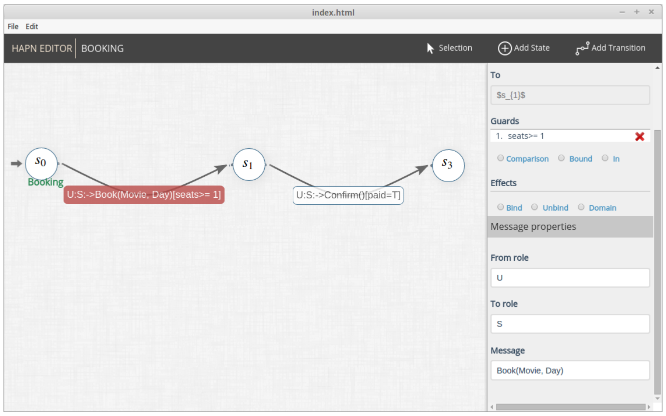

Figure 2: Transition properties in HAPN-E.

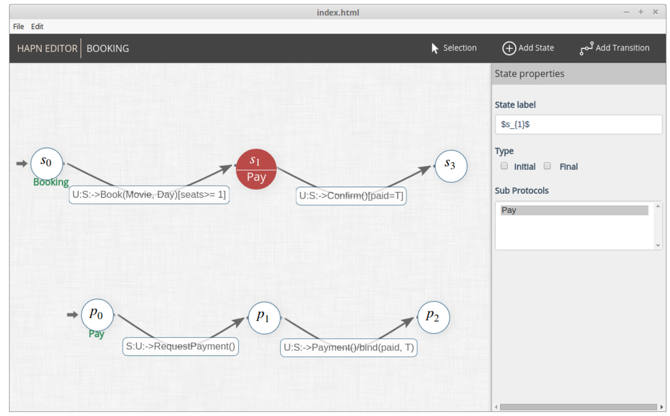

Figure 3: Assigning sub-protocols to a state in HAPN-E.

ure 1. In addition, we allow usage of LaTeX math mode to render state and protocol names. For example, s_0 in the state name field is visible as s_0 in the canvas.

2.2 Transitions

A transition can be added between two states s_0 and s_1 by clicking state s_0, dragging the mouse while pressed, and releasing on state s_1. HAPN-E provides a visual feedback while dragging the mouse from a state to another state. Releasing the mouse on the same state creates a loop transition on that state. Clicking an existing transition shows its property form (see Figure 2), which is used to modify the transition's message, guards, and effects.

Figure 2 shows a (incomplete) booking protocol. The transition $\mathsf{U} \to \mathsf{S} : \mathsf{Book}(Movie, Day)[seats >= 1]$ from state s_0 to s_1 contains message $\mathsf{Book}(Movie, Day)$, between roles user (U) and system (S), conditioned on availability of seats (captured by guard $[seats >= 1]$).

2.3 Sub-protocols

In order to facilitate modularity and re-use of protocols, we extend FSM's (in a fairly standard way [1]) into *hierarchical* FSMs. The basic idea is that each state can (optionally) contain a number of sub-protocols. A designer can associate sub-protocols to a state by selecting protocol names in the state's property form. In the editor, sub-protocols that are assigned to a state are shown below the state's name. Figure 3 shows Pay protocol assigned to state s_1 of the Booking protocol.

Conceptually, all sub-protocols in an active current state are executed in parallel. Hence, the set of acceptable messages in a current state that has sub-protocols will include acceptable message from the top level state itself along with acceptable messages from the sub-protocols (depending on their current state).

The graphical editor automatically maintains a protocol specification (i.e., its states and transitions) based on how the user links states with transitions.

2.4 Runtime execution

In addition to serving as a tool for designing protocols based on HAPN, the editor allows a protocol designer to simulate a protocol execution. Starting from the initial state of the top level protocol, the editor keeps a track of the current state and allows a user to *run* the protocol step by step by choosing the next transition at each step. This type of simulated runtime execution provides a designer important feedback such as the sequence of accepted messages and how relevant variables are updated.

2.5 Editor technology

The HAPN editor is built using HTML5, Javascript, and CSS and can be deployed both as a client/server and desktop based application. Integration with other desktop applications, such as model checkers and multi-agent software design tools, can be achieved by packaging HAPN-E using the node-webkit platform[1]. In our context, using web based technologies provides two key advantages: (i) it allows building upon existing Javascript visualisation libraries, and (ii) it enables us to share visualisation source code across web and desktop deployments.

3. FUTURE WORK

Currently we are working towards further refinement of the framework as well as enhancements of the graphical tool. An important next step is to assess comprehensibility of HAPN, this cannot be done objectively and will require experimental empirical evaluation. Other directions for future work include incorporating formal verification (such as checking of dead lock conditions) and integrating with existing agent design tools such as PDT [4]. Finally, we would like to develop a mapping to allow protocol execution monitoring, along the lines of Poutakidis *et al.* [3, 2].

Acknowledgments

This work is partially supported by the Australian Research Council and Real Thing Entertainment Pty. Ltd. under Linkage grant number LP110100050.

REFERENCES

[1] R. Alur. Formal analysis of hierarchical state machines. In N. Dershowitz, editor, *Verification: Theory and Practice*, volume 2772 of *Lecture Notes in Computer Science*, pages 42–66. Springer Berlin Heidelberg, 2003.

[2] L. Padgham, M. Winikoff, and D. Poutakidis. Adding debugging support to the Prometheus methodology. *Engineering Applications of Artificial Intelligence, special issue on Agent-oriented Software Development*, 18(2):173–190, 2005.

[3] D. Poutakidis, L. Padgham, and M. Winikoff. Debugging multi-agent systems using design artifacts: The case of interaction protocols. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 960–967, New York, NY, USA, 2002. ACM.

[4] J. Thangarajah, L. Padgham, and M. Winikoff. Prometheus design tool. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 127–128, New York, NY, USA, 2005. ACM.

[1] https://github.com/nwjs/nw.js/